**Capstone Project: Building a Command-Line To-Do List Application in Rust**

**Overview**

Welcome to your capstone project! In this project, you will build a simple command-line to-do list application using the Rust programming language. This project will give you hands-on experience with:

* Data Structures: Defining and manipulating custom structs.
* File I/O: Reading from and writing to JSON files to persist data.
* Command-Line Argument Parsing: Using clap to handle user input.
* Error Handling: Implementing robust error handling to manage invalid inputs and file operations.

By the end of this project, you will have a working command-line application that allows users to manage their to-do lists with tasks being saved to a file for persistence across program executions.

**Project Structure**

Your project will be organized into the following components:

* Task Management:
  + A Task struct that represents a task, including its ID, description, and completion status.
* File Handling:
  + Functions to read from and write to a JSON file where the tasks will be saved and loaded.
* Command-Line Argument Parsing:
  + Use the clap library to handle commands such as adding a task, listing tasks, and marking tasks as completed.
* Error Handling:
  + Proper error handling to manage cases where tasks cannot be loaded or saved due to file issues or invalid user input.

**Functional Requirements**

To successfully complete this project, ensure that your application meets the following functional requirements:

1. Task Management

* Task Struct:
  + Define a struct called Task that includes:
    - A unique id for each task (starting from 1 and incrementing with each new task).
    - A description for the task.
    - A completed field to indicate if the task has been finished.

2. Command-Line Operations

* Add Task:
  + Users should be able to add a task by providing a description.
  + Example command: todo add "Buy groceries"
* View Tasks:
  + Users should be able to list all tasks with their ID and status (completed or pending).
  + Example command: todo list
* Complete Task:
  + Users should be able to mark a task as completed by its ID.
  + Example command: todo complete 2

3. File Persistence

* Data Storage:
  + The application should save tasks to a file (tasks.json) in the local directory.
  + Tasks should be loaded from this file when the application starts so that the to-do list persists between runs.
* Serialization:
  + Use the serde and serde\_json crates to serialize the Vec<Task> into JSON format and write it to the file.
  + Deserialize the JSON file back into a Vec<Task> when loading tasks.

4. Error Handling

* File Handling:
  + Ensure that the application handles cases where the file cannot be read or written to, providing clear error messages to the user.
* Invalid Input:
  + Handle cases where users provide invalid task IDs or incomplete commands, offering helpful feedback.

**Supporting Documentation**

* README File:
  + Offer an overview of the project, the objectives, and the components.
  + Provide instructions on how to run the application, including the commands to add, list, and complete tasks.

**Evaluation Criteria**

Your project will be evaluated based on the following criteria:

* Completeness: All required features (add, list, complete) are implemented and function correctly.
* Correctness: The application correctly handles task management, file I/O, and error cases.
* Code Quality: The code is clean, well-organized, and follows Rust best practices.
* Documentation: The README file clearly explains how to use the application.

**Getting Started**

Step 1: Set Up the Project

* Initialize a new Rust project using Cargo:

bash

Copy code

cargo new todo

cd todo

Step 2: Implement Task Struct

* Define the Task struct that will hold the task data (ID, description, and completion status).

Step 3: Implement File Handling

* Write functions to load tasks from and save tasks to tasks.json using serde\_json.

Step 4: Parse Command-Line Arguments

* Use the clap library to parse commands such as add, list, and complete.

Step 5: Add Error Handling

* Ensure your program provides helpful feedback when errors occur, such as invalid input or file issues.

Step 6: Test the Application

* Test each feature of your application by running it through the command line:
  + Add tasks.
  + List tasks.
  + Complete tasks.

**Submission Guidelines**

* Repository: Host your project on GitHub or a similar platform.
* Files: Include the Rust source code and a README file explaining how to use the application.
* Instructions: Ensure that your README provides clear instructions on how to run your project.

**Conclusion**

This capstone project is a great opportunity to solidify your understanding of Rust while building a practical command-line tool. You’ll gain experience with data structures, file handling, error management, and command-line interfaces. Good luck, and enjoy building your to-do list manager!

**Resources:**

**Key Code:** [**https://github.com/mnechromancer/RustProjects/blob/main/todo/src/main.rs**](https://github.com/mnechromancer/RustProjects/blob/main/todo/src/main.rs)